

A Brief Overview of the Concepts of CEP¹ by David Luckham

The demand for processing higher level events, often called business events, has expanded rapidly over the past three or four years. In writing a short history of the development of CEP to meet the demands of these new markets, I have needed to refer often to a set of concepts that forms the core of CEP. For example, the question arises as to how much CEP is contained in the current CEP tools and applications out there in the marketplace. This is a brief and **incomplete** overview of some of those concepts.

Here's a short list of concepts:

- Events.
- relationships between events, such as timing, causality, independence, and similarity.
- patterns of events involving both sets of events and relationships between events.
- event pattern filters and constraints.
- event pattern triggered rules.
- complex events.
- event pattern abstraction.
- constructive hierarchies of events.

While most of these concepts are now well understood, there are many challenging research issues involved in developing efficient implementations to apply them to practical problems.

1. Events. An event is an object that represents or records an activity that happens, or is thought of as happening. Examples are:

- a purchase order (records a purchase activity),
- an email confirmation of an airline reservation,
- stock tick message that reports a stock trade,
- a message that reports an RFID sensor reading,
- a tuple that represents a change of direction in a simulation of an airplane.

The last example is a virtual event – one that is thought of as happening.

¹ © 2007 David C Luckham

In event processing we do not distinguish between an *activity* that takes place and an *object* that represents that activity for the purpose of computer processing. The word “event” is overloaded to take both meanings.

2. Patterns of events.

There are many differing ideas about patterns of events. For some, an event pattern is simply a Boolean combination of events, like A **and** B. For others it is an SQL query. But in its most general form in CEP, an event pattern can be a timing or causal relationship between patterns of events as well as a Boolean combination. A pattern contains variables which are replaced by values to form instances of the pattern. A pattern can have many different instances. Examples are

- Shopping (?Customer, ?Time, ?DollarAmount)
- $A \rightarrow B$
- A **at** t **and** B **at** t'
- A **at** t \rightarrow B **at** t'
- A **within** [t, t']

In the first example of an event pattern the variables are preceded by “?”. Its instances are Shopping events with actual customer, time and dollar values. In other examples “ \rightarrow ” means “causes”, [t, t'] is an interval between two times t, t', and A and B are themselves patterns of events. Relationships between events such as causality and time are an integral part of patterns of events in CEP.

Event patterns involve several technologies:

- *Event pattern language design.* The design of languages in which you can define the patterns you want to detect. Language design deals with questions such as the kinds of event relationships and combinations of events that may be used in a pattern, how the variable parts of the pattern are expressed, whether arbitrary *sequences* of events and pattern abstraction are allowed.
- *Semantics of event pattern matching.* An event pattern language must have a precise specification of the concept of a pattern match. This is how the semantics of the pattern language is defined. It is fundamental to ensuring that patterns are written correctly, and express the writer's intentions. Not only that, but the implementations of pattern matching by event pattern engines must be correct — an area that receives scant attention at the moment.

- *Analysis of pattern detection results:* the results of pattern searching must be presented in a way that is understandable and enables further actions or decisions to be taken, either by humans or by automated processes. The most popular technologies for this at the moment are the graphical dashboard and reactive rules.

Event pattern definition and matching is a fundamental part of CEP. It involves a lot of new design and implementation details in building efficient pattern matchers that scale to high speed event flows or can detect patterns of multiple events occurring over long periods of time. This technology is still in its infancy. For example, no commercial event pattern language includes an operator to express causality at the moment.²

Here are some examples of event pattern matching that have seen recent commercial application:

1. Looking for what you expect to find in the event cloud: Pretty much any activity in the enterprise is signified by a *pattern of events*. But of course all the different activities are jumbled up in the same cloud, so we have to sort them out.

For example, sales at an on-line website involve sequences of events where a customer logs onto the site, chooses items, puts them in a shopping cart, and goes through a checkout process. Essentially this is a sequence of several events, repeated over and over by different customers buying different items. In fact we don't know in advance how many events will be in the sequence because we don't know what each customer is going to buy. All of the possible event sequences can be represented by a pattern in which the customers, items, and the actual number of events are variables. The pattern language has to be powerful enough to express this kind of pattern.

2. Detecting the unexpected in the event cloud. This is a harder problem since it is difficult to define event patterns you don't expect! But it is just as important as the first problem, sometimes much more important. Let's take a couple of simple examples.

First of all, consider the on-line website shopping cart activity again. Customers often drop the cart right in the middle of shopping and walk away. Enterprises in on-line retail are often as interested in detecting the dropped cart behavior as they are in completed sales. They want to know immediately the drop rates increase above average. And then they want to know why. What can be done?

One approach is to use time windows in defining the expected shopping pattern. If the pattern fails to complete a match within the time window, a warning is triggered that perhaps there is a dropped cart. The pattern can also refer to customer history in defining the time window. In fact the real issue here is event

² June 2008.

pattern languages design. To write these kinds of patterns so you don't get too many false warnings, the language has to be able to express time windows, customer history and so on.

Another example is security. Credit card companies want to detect patterns that might indicate someone using a stolen card. You can write patterns that are indicative of instances of this, for example if a card is used in widely separate locations within a short time interval, or if the cardholder's spending pattern suddenly changes. It is true that these patterns are usually discovered by bad experiences, but at least the card company won't get burned twice by the same scam!

Security defense also involves detecting activity in the event cloud that might be the result of processes that are not within the enterprise's control – spyware for example, possibly loaded by insiders. The first indication might be an activity where supposedly secure data is being mailed to some outside website – this kind of activity should always be detected by any event traffic monitor on the cloud. It is basic protection.

2. Event pattern constraints

Another important concept in CEP is *event pattern constraints*. A constraint expresses a condition that must be satisfied by the activities of an enterprise.

- *Event pattern constraints: An event pattern constraint is a pattern that should never occur in the activity of the enterprise.*

Essentially, a constraint is a *negation* of an event pattern. In the day to day running of the modern enterprise knowing that its activities are not violating its guidelines, requirements or company policies, is often more important than knowing exactly the details of that activity. Here are some examples.

3. Keeping the enterprise within its policy boundaries.

An operator of a call center will have service level agreements (SLAs) with its clients. Examples:

- All callers must be serviced within an average of X minutes wait time.
- If a caller's issue cannot be resolved there must be follow up correspondence with that caller within Y hours.

The first constraint expresses that the average wait time for service must never exceed X minutes. If the average wait time exceeds X minutes, the constraint is violated. Then the warning bells start to ring! And the operator has to fix the wait

time. In the second case, if at the end of a call there is an unresolved issue, then a constraint is triggered and the clock starts ticking. If that constraint is not matched within Y hours, it is violated and further action must be taken.

Constraints can be implemented by event pattern triggered rules. The pattern triggers are run continuously against the cloud of event traffic coming from the call center. If those patterns ever match, then the rules are triggered and violation alerts issued.

Policies are another example of constraints that the event cloud should satisfy. No better examples can be found than in financial trading. There is the famous example of Nick Leeson who bankrupted Barings bank by trading on the Japanese markets in amounts well beyond his authorizations, and the bank's abilities to pay.³ Or the recent Societe Generale trader, Jerome Kerviel, whose market bets well exceeded his permissions⁴. The second case is ongoing, but it is unbelievable that a banking enterprise today would not have the technology to monitor its own data bases in real time for permission violations or fictitious trades.

Policy conformance is a market opportunity that the CEP vendors are beginning to exploit.

3. Event pattern triggered rules

The use of reactive rules is well known in other areas of computer science. The only new thing in CEP is the use of event patterns to trigger reactive rules and take actions when those patterns occur. This is the state of the art in commercial BAM today. In the first BAM applications pattern triggers were often just single events. But this situation is changing with the latest generation of BAM products.

Example:

- **Whenever** IBM trades 2% above its 1 hour VWAP **and** within 15 minutes trades 5 points below that 1 hr VWAP
then buy 1000 shares IBM at any price below that 1hr VWAP.

In this example the variable "1 hr VWAP" is bound to a value in an instance of the trigger pattern and used in the resulting action.

4. Complex event: an event that is an abstraction of other events called its *members*.

³ This event is so famous that Leeson is now in Wikipedia, along with various Nobel prize winners
http://en.wikipedia.org/wiki/Nick_Leeson

⁴ <http://www.forbes.com/feeds/ap/2008/02/06/ap4623008.html>

Examples:

- the 1929 stock market crash (an abstraction denoting many thousands of member events, including individual stock trades),
- the 2004 Indonesian Tsunami (an abstraction of many natural events) ,
- a CPU instruction (an abstraction of register transfer level (RTL) events),
- a completed stock purchase (an abstraction of the events in a transaction to purchase the stock).

A complex event *denotes* or *signifies* the set of its member events. It is tempting to say that a complex event “references” the set of its members, the implication being that the event contains a reference. In many cases this is true. But in general “reference” may be too strong a requirement since it implies that the members can be determined from the reference, e.g., there is no accepted agreement as to which events are members of the 1929 stock market crash.

5. Event pattern abstraction

Using a complex event to summarize the data contained in a pattern of events is a featured technique in CEP.

- An event is an *abstraction* of a pattern of events if it summarizes, represents, or denotes that set of events.

In practice event pattern abstraction is often defined by pattern triggered rules. When the pattern trigger matches, the rule creates a new event that abstracts data in the instance that matched the pattern.

In the on-line website operation, for example, each successful customer visit is a sequence of events where a customer logs onto the site, chooses items, puts them in a shopping cart, and then completes a checkout process. For different sales the sequences are different and contain different numbers of events. But each successful visit could be abstracted by a single event, “successful visit” containing just the data the retailer might want to keep: e.g., customer name, date, amount of sale, length of time of the visit, and traffic density (i.e., the number of concurrent visits in progress).

The complementary concept to pattern abstraction is *drill down*.

- Given an abstract event it should be possible to recall the set of events that it abstracts.

Drill down is not always possible (e.g., the 1929 crash). But when abstractions are created by means of rules, then the instances of the triggers can be retained in an archive to allow drill down.

For example, given a successful sale event, we can recall the sequence of events of that particular visit. Note that two identical successful sale events (i.e., same customer, items, sale amount, but at different times) might well abstract different sequences of website activity. Perhaps on the first visit the customer chose and then returned some items, whereas on the second visit the customer knew exactly what was wanted.

We don't often think of some popular kinds of event processing as computing abstractions. But in fact they usually are. For example a market trading application shown in figure 1 is taking as input two stock market feeds and

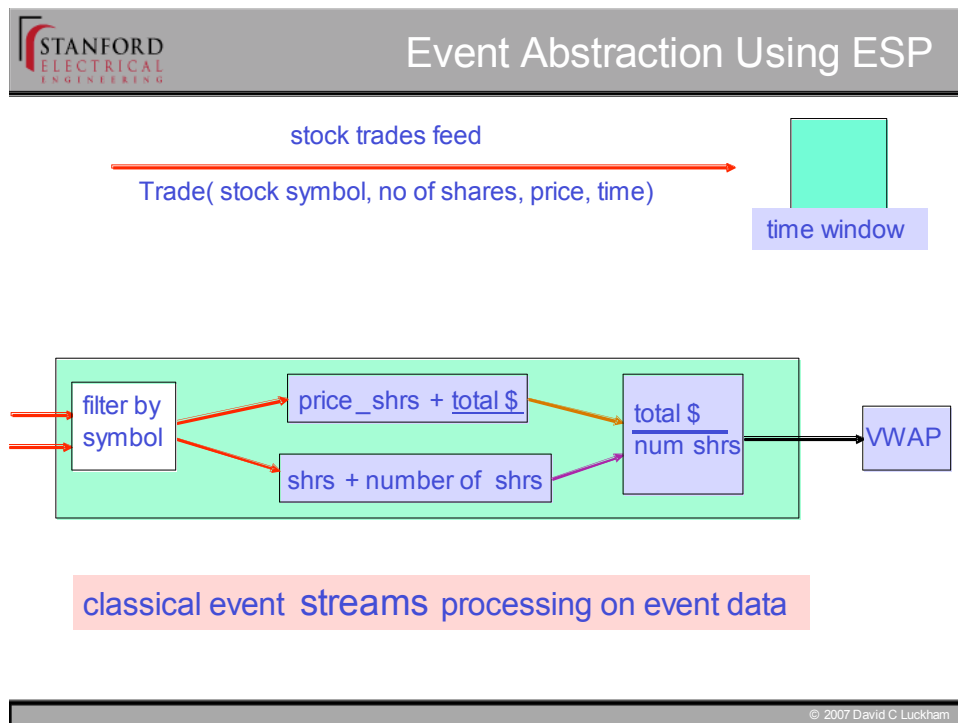


Figure 1: abstracting stock market feeds by VWAP

computing the volume weighted average price of stocks over a time window. The inputs are streams of events and the output is a stream of VWAPs. Each VWAP abstracts the data contained in the trading events during a time window.

5. hierarchies of events

Hierarchical structure is ubiquitous in everyday life. In CEP event hierarchies are a computable method of organizing event abstractions into levels.

An *event abstraction hierarchy* consists of the following elements.

- *A sequence of levels of activities and associated event types.* Each level consists of a set of descriptions of system activities and, for each activity, a specification of the types of events that signify instances of that activity. Level 1 is the lowest level.
- *A set of event abstraction rules for each level.* For each level (except level 1), there must be a rule for creating each type of event at that level as an abstraction of patterns of events at levels below.

So an event hierarchy defines a set of levels of activities and a set of rules for computing events at each level as abstractions of patterns of events from the levels below.

The crucial aspect of this definition is the set of rules specifying how each event at a higher level is an abstraction of events at levels below it. This is where a CEP event hierarchy goes beyond prior uses of hierarchies in system organization. A hierarchy in CEP gives a constructive method – e.g., rules – for computing abstract events from events that can be observed in a system.

The retail website example can provide lots of event hierarchies. A three level hierarchy could be the following.

- The **lowest level** events might be the operations in the website: accepting a new visitor, creating a view for that visitor, making entries into the data base, updating a view, and so on.
- At a level above, which we might call the **website viewing level**, we might define events such as successful visits, dropped carts, traffic density, etc. Finally we would define the rules for creating events such as successful visits or dropped carts from patterns of events at the website operations level.
- A higher level might be the **sales promotions results view** which would combine events at the viewing level with sales promotion events such as reduced prices using time windows to relate events.

An important point is *flexibility* in hierarchy definitions. Event hierarchies can be changed, new levels of events inserted, or levels removed. One must make the corresponding modifications to the abstraction rules to support any change to the event levels.

Each problem domain, say on-line website operations, will give rise to event hierarchies to organize the processing of events in that domain. On-line websites, call centers, factory fabrication lines, will all create different types of

events and will need different kinds of event processing in their operation. Different operators will use their own personalized event hierarchies.

Sometimes, an event hierarchy will become a standard. It happens when the enterprises concerned with a specific type of problem domain decide to consolidate their efforts. In the past hierarchy standards have happened in networking and in hardware design.

As far as I can determine, nothing in the direction of defining event hierarchies, or documenting their use, is taking place in business processing. This will have to happen in the future. Standards in event processing are a precondition to sharing event processing tools easily, or even having an understandable dialogue about the results and experiences with different applications.